

# An exact algorithm and a heuristic for scheduling linearly deteriorating jobs with arbitrary precedence constraints and the maximum cost criterion

Marek Dębczyński and Stanisław Gawiejnowicz

Adam Mickiewicz University in Poznań  
 Faculty of Mathematics and Computer Science  
 Umultowska 87, 61-614 Poznań, Poland  
 Email: mdeb@amu.edu.pl, stgawiej@amu.edu.pl

**Abstract**—We consider the problem of scheduling linearly deteriorating jobs on a single machine. Between the jobs there are defined arbitrary precedence constraints and the objective is to minimize the maximum cost. For this problem we propose an exact algorithm and a heuristic. We also report preliminary results of computational experiments conducted in order to evaluate the quality of schedules generated by the heuristic.

## I. INTRODUCTION

**S**CHEDULING jobs with variable processing times very often occur in manufacturing practice [6]. For example, the processing time of an activity may be shorter (longer) if we pay more (less) for its completion. Hence, managers and project leaders must control the processing times of activities in projects they rule, in order to act accordingly to changes in the projects.

In scheduling theory there are known a few different models of variable processing times. One of them is *time-dependent scheduling* in which one assumes that the processing time of a job is a function which depends on the starting time of the job. This assumption allows us to consider such scheduling problems in which any delay results in changes of job processing times. We refer the reader to [1] for the most recent discussion on different aspects of time-dependent scheduling.

Among many time-dependent scheduling problems considered in literature, most often are studied problems in which job processing times are described by monotonically increasing functions. So defined jobs are called deteriorating jobs, since the job processing times *deteriorate* (increase) in time.

Usually deteriorating jobs are assumed to be independent, i.e. no precedence constraints exist between the jobs. This is a significant simplification, since in real-life problems precedence constraints occur very often. For example, project managers must take into account not only variable processing times of activities but also precedence constraints that indicate which activities must be completed in order to start other ones. Hence, even single machine scheduling problems with the maximum completion time criterion ( $C_{\max}$ ), variable job processing times and non-empty precedence constraints are rather weakly explored, since only a few results are known. A special case of linearly deteriorating jobs with arbitrary

precedence constraints is considered in [4]. Linear jobs with a few special forms of precedence constraints are discussed in [7] and [1, Chapter 13].

In this paper, we consider the following problem of scheduling dependent deteriorating jobs on a single machine. The processing time of each job is a linear function of the job starting time and between the jobs there exist arbitrary precedence constraints described by a directed acyclic graph. The criterion of optimality of a schedule is to minimize the maximum cost. The criterion is a generalization of the  $C_{\max}$  criterion and in a better way mirrors the criteria applied in real-life problems. For example, if a company pays after completion of each activity of a project and it uses a bounded budget, the manager will look for such a schedule in which the maximum single payment will be minimized. According to the best of our knowledge, scheduling problems of this type were not considered in literature earlier.

The remaining sections of this paper are organized as follows. In Section 2, we formulate the considered problem. In Section 3, we formulate the exact algorithm. In Section 4, we present the heuristic algorithm. In Section 5, we present the results of a numerical experiment. In Section 6, we give conclusions and remarks for further research.

## II. PROBLEM FORMULATION

The problem under consideration can be formulated as follows. The set of jobs  $\mathcal{J} = \{J_1, J_2, \dots, J_n\}$  has to be processed on a single machine, available for processing from time  $t_0 \geq 0$ . Job processing time of job  $J_j \in \mathcal{J}$  at time  $t \geq t_0$  equals  $p_j(t) = a_j + b_j * t$ , where basic job processing times  $a_j \geq 0$  and deterioration rate  $b_j \geq 0$  for  $1 \leq j \leq n$ . Each job  $J_j \in \mathcal{J}$  is associated with a monotone non-decreasing cost function  $f_j(t)$ . It measures the cost of the completion of job  $J_j$  at time  $t$  and can be computed in a constant time. Job precedence constraints between jobs from the set  $\mathcal{J}$  are described by a given acyclic digraph  $G = (V(G), A(G))$ , where  $V(G)$  and  $A(G)$  denote the set of vertices and the set of arcs, respectively. Our goal is to find a schedule  $\sigma$  such that  $f_{\max}(\sigma) = \min_{\tau \in \mathcal{S}} \max_{1 \leq j \leq n} \{f_j(C_j(\tau))\}$ , where  $C_j(\tau)$  and  $S$  denote the completion time of job  $J_j \in \mathcal{J}$  in the schedule

$\tau$  and the set of all schedules consistent with job precedence constraints described by  $G$ , respectively.

In short, we will call the problem as the SPDJMS (Single-machine Precedence-constrained Deteriorating Jobs Maximum-cost Scheduling) problem.

A special case of the SPDJMS problem is the problem of minimizing the maximum lateness for a set of jobs with mixed job processing times and empty precedence constraints, proved to be *NP*-hard in [2]. Therefore, the problem under consideration is *NP*-hard as well and for it there do not exist polynomial algorithms, unless  $P = NP$ .

Since the SPDJMS problem is *NP*-hard, only suboptimal schedules for it can be constructed in polynomial time. Hence, we are interested in good polynomial heuristic algorithms for the problem. In our paper, we propose such an algorithm. The quality of schedules generated by the algorithm was examined by using an exact algorithm described in the next section.

### III. EXACT ALGORITHM

In the section, we formulate an exact algorithm for our problem. The algorithm will be used to verify the quality of schedules generated by a heuristic algorithm for the problem presented in the next section. Before we formulate the exact algorithm, we define two auxiliary functions: GENERATEALLTOPSORTS and CALCULATECT.

The function GENERATEALLTOPSORTS applies the idea used in the algorithm presented in [3]. Given a digraph  $G$  the algorithm generates all topological sortings and returns them as a list  $H$  of dimension SIZEOF( $H$ ). The algorithm uses the following data structures: a list VERTICES, an array PREDSCOUNT and a list VEP, which contain vertices, number of predecessors for each vertex and vertices without predecessors, respectively. For a given  $v \in V(G)$ , attribute PredCount contains the number of predecessors of  $v$ , while method Add inserts a vertex to the list VEP.

```

1: function GENERATEALLTOPSORTS( $G$ )
2:   for all  $v$  in  $V(G)$  do
3:     VERTICES[ $v$ .Index]  $\leftarrow v$ 
4:     PREDSCOUNT[ $v$ .Index]  $\leftarrow v$ .PredCount
5:     if PREDSCOUNT[ $v$ .Index] = 0 then
6:       VEP.Add( $v$ .Index)
7:   end for
8:   ALLTOPSORT(0)
9:   return  $H$ 

```

The function GENERATEALLTOPSORTS after creating mentioned data structures, calls the function ALLTOPSORT.

The function ALLTOPSORT returns all topological sortings as a list  $H$ . It manipulates the list VEP which contains only those vertices without predecessors that have not been yet processed. The function ALLTOPSORT may cause temporary changes in VEP and PREDSCOUNT, but upon exit they are restored to initial values. Functions ERASEALLRELATIONS( $q$ ) and RETRIEVEALLRELATIONS( $q$ ) erase and retrieve all arcs  $(q, i) \in A(G)$  for a given vertex  $q \in V(G)$ , respectively. The array OUTPUT contains a sequence of already processed

vertices, and if its size is equal to the number of vertices of  $G$ , the sequence is moved to the list  $H$ .

```

1: function ALLTOPSORT( $k$ )
2:   if (VEP.Count > 0) then
3:      $base \leftarrow$  VEP[VEP.Count]
4:     do
5:        $q \leftarrow$  VEP[VEP.Count]
6:       VEP  $\leftarrow$  VEP \ { $q$ };
7:       ERASEALLRELATIONS( $q$ )
8:       OUTPUT[ $k$ ]  $\leftarrow q$ ;
9:       if  $k =$  VERTICES.Count-1 then
10:         $H$ .Add(OUTPUT)
11:      ALLTOPSORT( $k + 1$ )
12:      RETRIEVEALLRELATIONS( $q$ )
13:      VEP.Insert(1, $q$ )
14:     while VEP[VEP.Count]  $\neq base$ 

```

*Example 1:* We illustrate the application of the algorithm to a digraph given in Fig. 1.

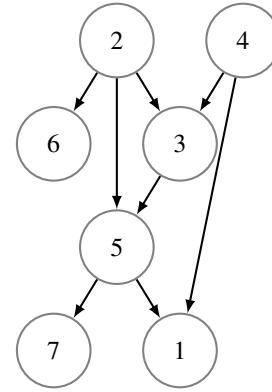


Fig. 1. Digraph of job precedence constraints

There exist 22 topological sortings of the digraph. Each sorting corresponds to a sequence of indices of vertices of the digraph that, in turn, corresponds to a schedule. All these schedules, denoted as  $\sigma_i$  for  $1 \leq i \leq 22$  (see Table I), can be found using the function GENERATEALLTOPSORTS.

TABLE I  
ALL TOPOLOGICAL SORTINGS FOR DIGRAPH IN FIG. 1

Schedule	Sorting	Schedule	Sorting
$\sigma_1$	(2,6,4,3,5,1,7)	$\sigma_{12}$	(2,4,6,3,5,7,1)
$\sigma_2$	(2,6,4,3,5,7,1)	$\sigma_{13}$	(4,2,6,3,5,1,7)
$\sigma_3$	(2,4,3,5,1,7,6)	$\sigma_{14}$	(4,2,6,3,5,7,1)
$\sigma_4$	(2,4,3,5,1,6,7)	$\sigma_{15}$	(4,2,3,5,1,7,6)
$\sigma_5$	(2,4,3,5,7,6,1)	$\sigma_{16}$	(4,2,3,5,1,6,7)
$\sigma_6$	(2,4,3,5,7,1,6)	$\sigma_{17}$	(4,2,3,5,7,6,1)
$\sigma_7$	(2,4,3,5,6,1,7)	$\sigma_{18}$	(4,2,3,5,7,1,6)
$\sigma_8$	(2,4,3,5,6,7,1)	$\sigma_{19}$	(4,2,3,5,6,1,7)
$\sigma_9$	(2,4,3,6,5,1,7)	$\sigma_{20}$	(4,2,3,5,6,7,1)
$\sigma_{10}$	(2,4,3,6,5,7,1)	$\sigma_{21}$	(4,2,3,6,5,1,7)
$\sigma_{11}$	(2,4,6,3,5,1,7)	$\sigma_{22}$	(4,2,3,6,5,7,1)

Now, we describe the function CALCULATECT, the main idea of our exact algorithm and we present its pseudo-code.

The function CALCULATECT, for a given schedule  $H[i]$  and starting time  $t_0$ , generates the vector of all job completion times. The pseudo-code of the function is as follows.

```

1: function CALCULATECT( $H[i], t_0$ )
2:    $C[0] \leftarrow t_0$ 
3:   for  $i \leftarrow 1$  to  $n$  do
4:      $C[i] \leftarrow a_i + (1 + b_i) * C[i - 1]$ 
5:   end for
6:   return  $C$ 

```

The main idea of our exact algorithm (see the pseudo-code of Algorithm 1) is as follows. Using function GENERATEALLTOPSORTS we create all topological sortings and store them in array  $H$ . Each such a sorting corresponds to a schedule and is described by an element  $H[i]$  from  $H$ . For each schedule  $\sigma$  described by a  $H[i]$ , we calculate job completion times by function CALCULATECT and store them in array  $C$ . For each job  $J_j$  we compute its cost in  $\sigma$  and  $C$ , and calculate the maximum cost  $F_{\max}$  of currently examined schedule. Finally, we compare the value  $F_{\max}$  with previously stored value of  $F_{\min}$  of the smallest maximum cost among all examined schedules and if  $F_{\max}$  is lower, we set  $F_{\min} \leftarrow F_{\max}$ . In a similar way, we check all remaining sortings in  $H$ .

At the input of Algorithm 1 we are given a digraph  $G$  of job precedence constraints, the set of jobs  $\mathcal{J}$  with processing times in the form of  $p_j(t) = a_j + b_j * t$  and cost functions  $f_j$ , where  $1 \leq j \leq n$ . At the output of the algorithm we get optimal schedule  $\sigma^*$  with the smallest maximum cost.

---

**Algorithm 1** for the SPDJMS problem

---

```

1:  $H \leftarrow \text{GENERATEALLTOPSORTS}(G)$ 
2:  $k \leftarrow \text{SIZEOF}(H)$ 
3:  $i \leftarrow 1$ 
4:  $F_{\min} \leftarrow \infty$ 
5: while  $i \leq k$  do
6:    $F_{\max} \leftarrow -\infty$ 
7:    $C \leftarrow \text{CALCULATECT}(H[i], t_0)$ 
8:    $r \leftarrow n$ 
9:   while  $r \geq 1$  do
10:     $j \leftarrow H[i][r]$ 
11:     $f \leftarrow f_j(C[r])$ 
12:    if  $F_{\max} < f$  then  $F_{\max} \leftarrow f$ 
13:     $r \leftarrow r - 1$ 
14:   end while
15:   if  $F_{\min} > F_{\max}$  and  $r = 0$  then
16:      $F_{\min} \leftarrow F_{\max}$ 
17:      $\sigma^* \leftarrow H[i]$ 
18:    $i \leftarrow i + 1$ 
19: end while
20: return  $\sigma^*$ 

```

---

#### IV. HEURISTIC ALGORITHM

In the section, we present a heuristic algorithm for the SPDJMS problem, based on algorithm from [5] for finding a schedule with the smallest maximum cost for a set of jobs with fixed job processing times. Before we formulate the heuristic algorithm, we define three auxiliary functions: CALCULATEFMAX, GENERATERATES and FINDMINIMUM.

The function CALCULATEFMAX, for a given schedule  $\sigma$  and starting time  $t_0$ , calculates the maximum cost  $F_{\max}$ .

```

1: function CALCULATEFMAX( $\sigma, t_0$ )
2:    $C[0] \leftarrow t_0$ 
3:    $F_{\max} \leftarrow -\infty$ 
4:   for all  $j$  in  $\sigma$  do
5:      $C[j] \leftarrow a_j + (1 + b_j) * C[j - 1]$ 
6:      $f \leftarrow f_j(C[j])$ 
7:     if  $F_{\max} < f$  then  $F_{\max} \leftarrow f$ 
8:   end for
9:   return  $F_{\max}$ 

```

The function GENERATERATES, for a given set of jobs  $\mathcal{J}$ , for each job returns a pair of two sets of ratios. The first set contains the values of  $\frac{b_j}{a_j}$  for each job  $J_j \in \mathcal{J}$ . We use the ratios to generate schedules with the smallest and the largest completion time of the last job. The second set contains the values of cost functions associated with each job  $J_j \in \mathcal{J}$  at time  $t = \text{Random}(1, 100)$ . We use this set to generate schedule with jobs in non-increasing order of these ratios.

```

1: function GENERATERATES( $\mathcal{J}$ )
2:   for all  $j$  in  $\mathcal{J}$  do
3:      $t \leftarrow \text{RANDOM}(1, 100)$ 
4:      $ptRates[j] \leftarrow \frac{b_j}{a_j}$ 
5:      $fRates[j] \leftarrow f_j(t)$ 
6:   end for
7:   return ( $ptRates, fRates$ )

```

The function FINDMINIMUM, using the function CALCULATEFMAX, for each of given four schedules  $\sigma_1, \sigma_2, \sigma_3, \sigma_4$ , calculates the maximum cost and returns both an optimal schedule and its maximum cost.

```

1: function FINDMINIMUM( $\sigma_1, \sigma_2, \sigma_3, \sigma_4, t_0$ )
2:   for  $i \leftarrow 1$  to 4 do
3:      $F[i] \leftarrow \text{CALCULATEFMAX}(\sigma_i, t_0)$ 
4:   end for
5:    $F_{\min} \leftarrow F[1]$ 
6:   for  $i \leftarrow 2$  to 4 do
7:     if  $F[i] < F_{\min}$  then
8:        $F_{\min} \leftarrow F[i]$ 
9:        $\sigma \leftarrow \sigma_i$ 
10:  end for
11:  return ( $\sigma, F_{\min}$ )

```

Now, we describe our heuristic algorithm (see the pseudo-code of Algorithm 2). At the input of the algorithm, we are given a digraph  $G$  of job precedence constraints, a set of jobs  $\mathcal{J}$  with processing times in the form of  $p_j(t) = a_j + b_j * t$  and cost functions  $f_j$ ,  $1 \leq j \leq n$ . By  $G_T, V(G_T), A(G_T)$

and  $NS(G_T)$  we denote the copy of digraph  $G$  of precedence constraints, the set of vertices of  $G_T$ , the set of arcs in  $G_T$  and the set of jobs without successors in  $G_T$ , respectively. At the output of the algorithm, we get an optimal schedule  $\sigma^*$ .

---

**Algorithm 2** for the SPDJMS problem
 

---

```

1: for  $i \leftarrow 1$  to 4 do
2:    $G_T \leftarrow G$ 
3:   while  $V(G_T) \neq \emptyset$  do
4:      $NS(G_T) \leftarrow \{j \in V(G_T) : \text{deg}^+(j) = 0\}$ 
5:      $\min \leftarrow \infty$ 
6:      $\max \leftarrow -\infty$ 
7:     GENERATERATES( $\mathcal{J}$ )
8:     for all  $j$  in  $NS(G_T)$  do
9:       if  $i = 1$  and  $ptRates[j] < \min$  then
10:         $\min \leftarrow ptRates[j]$ 
11:         $k \leftarrow j$ 
12:       else if  $i = 2$  and  $ptRates[j] > \max$  then
13:         $\max \leftarrow ptRates[j]$ 
14:         $k \leftarrow j$ 
15:       else if  $i = 3$  and  $fRates[j] > \max$  then
16:         $\max \leftarrow fRates[j]$ 
17:         $k \leftarrow j$ 
18:     end for
19:     if  $i = 1$  then  $\sigma_1 \leftarrow (k|\sigma_1)$ 
20:     else if  $i = 2$  then  $\sigma_2 \leftarrow (k|\sigma_2)$ 
21:     else if  $i = 3$  then  $\sigma_3 \leftarrow (k|\sigma_3)$ 
22:     else
23:        $x \leftarrow \text{SIZEOF}(NS(G_T))$ 
24:        $k \leftarrow x \text{ div } \text{Random}(1, x)$ 
25:        $\sigma_4 \leftarrow (NS(G_T, k) | \sigma_4)$ 
26:        $V(G_T) \leftarrow V(G_T) \setminus \{k\}$ 
27:        $A(G_T) \leftarrow A(G_T) \setminus \{(i, k) : i \in V(G_T)\}$ 
28:     end while
29: end for
30:  $(\sigma^*, F_{\min}) \leftarrow \text{FINDMINIMUM}(\sigma_1, \sigma_2, \sigma_3, \sigma_4, t_0)$ 
31: return  $\sigma^*, F_{\min}$ 

```

---

The main idea of Algorithm 2 is as follows. Four iterations of the external loop **for** (lines 1–29) are responsible for creating four different schedules  $\sigma_1, \sigma_2, \sigma_3, \sigma_4$ . Schedules  $\sigma_1$  and  $\sigma_2$  correspond to feasible schedules with the smallest and the largest maximum completion time, respectively. Schedule  $\sigma_3$  corresponds to non-increasing order of values of cost functions calculated by function GENERATERATES. The last schedule  $\sigma_4$  is a random schedule, which does not violate job precedence constraints. In every iteration of the internal loop (lines 3–28) a set of jobs without successors (line 4) is created, and for this set schedules  $\sigma_1, \sigma_2, \sigma_3, \sigma_4$  (lines 7–27) are constructed. In every iteration of loop **for** (lines 8–18) there is selected a job to schedule  $\sigma_1$  (lines 9–11),  $\sigma_2$  (lines 12–14) and  $\sigma_3$  (lines 15–17). Conditional statements **if** in lines 19–21 are responsible for adding the chosen job to an appropriate schedule ( $\sigma_1, \sigma_2$  or  $\sigma_3$ ), while in lines 22–25 we choose a random job from the set  $NS(G_T)$  and add it to schedule  $\sigma_4$ .

Next, in lines 26–27, we update sets  $V(G_T)$  and  $A(G_T)$ . We proceed in a similar way with other jobs until the moment when  $V(G_T) = \emptyset$ .

*Example 2:* We apply the Algorithm 2 to the following instance. We are given a set of  $n = 7$  jobs, with processing times and cost functions as in Table II. A digraph of precedence constraints between the jobs is given in Fig. 1.

TABLE II  
JOB PROCESSING TIMES AND COST FUNCTIONS IN EXAMPLE 2

Processing times	Cost functions
$p_1(t) = 8 + 3t$	$f_1(t) = t + 5$
$p_2(t) = 4 + t$	$f_2(t) = 2t + 3$
$p_3(t) = 1 + t$	$f_3(t) = t + 4$
$p_4(t) = 7 + 2t$	$f_4(t) = 2t + 8$
$p_5(t) = 3 + 2t$	$f_5(t) = 2t + 7$
$p_6(t) = 8 + 2t$	$f_6(t) = t + 2$
$p_7(t) = 2 + t$	$f_7(t) = 2t + 6$

For this instance, the heuristic algorithm generates sequence  $\sigma_{18}$  with the maximum cost equal to 2794. All possible schedules for the instance, together with corresponding maximal costs, are listed in Table III (by a star '\*' we denote the optimal schedule and its maximum cost).

TABLE III  
ALL FEASIBLE SCHEDULES FOR INSTANCE FROM EXAMPLE 2

Schedule	Maximum cost	Schedule	Maximum cost
$\sigma_1$	6 570	$\sigma_{12}$	3 189
$\sigma_2$	3 285	$\sigma_{13}$	6 090
$\sigma_3$	2 944	$\sigma_{14}$	3 045
$\sigma_4$	5 898	$\sigma_{15}$	2 800
$\sigma_5$	2 949	$\sigma_{16}$	5 610
$\sigma_6$	2 938	$\sigma_{17}$	2 805
$\sigma_7$	5 930	$\sigma_{18}^*$	2 794*
$\sigma_8$	2 965	$\sigma_{19}$	5 642
$\sigma_9$	6 090	$\sigma_{20}$	2 821
$\sigma_{10}$	3 045	$\sigma_{21}$	5 802
$\sigma_{11}$	6 378	$\sigma_{22}$	2 901

## V. NUMERICAL EXPERIMENT

In the section, we present results of a numerical experiment we conducted in order to verify the quality of schedules constructed by the proposed heuristic algorithm.

We implemented both the exact algorithm and the heuristic algorithm in Microsoft Visual C# 2010 Express. The experiment was performed on a PC computer with the following parameters:

- 1) Motherboard: ASUS P5K SE;
- 2) Processor: Intel(R) Pentium(R) Dual CPU E2180 @ 2.00GHz (2C 2GHz, 1MB L2);
- 3) RAM Memory: 2GB of Patriot Memory DDR2-800;
- 4) Operating System: Microsoft Windows XP Professional.

In the experiment, we generated 6 sets with  $n$  jobs, where  $n \in \{5, 6, 7, 8, 9, 10\}$ . For each  $n$ , we generated 10 times a random digraph  $G$  of job precedence constraints. For each such a digraph, we calculated the smallest maximum cost and an optimal schedule using the exact algorithm and a suboptimal schedule and its maximum cost using the heuristic. In total, we generated 60 random instances of our problem.

Results of the experiment are summarized in Table IV, where '#sched', 'Exact', 'Heur' and 'Worst' denote the number of all feasible schedules, the optimal (minimal) maximum cost, the maximum cost of the schedule constructed by our heuristic and the worst (largest) maximum cost, respectively. Each value in the table is an average of 10 values corresponding to randomly generated instances of the SPDJMS problem.

Average computation time for the heuristic algorithm was less than 1ms. Average time for the exact algorithm for instances with  $n = 10$  jobs was less than 10s.

TABLE IV  
RESULTS OF NUMERICAL EXPERIMENT

$n$	#sched	Exact	Heur	Worst
5	15	546	602	1 297
6	68	1 516	2 078	5 525
7	105	6 463	7 857	25 769
8	882	10 917	14 631	50 788
9	5 851	9 385	13 667	76 127
10	54 564	51 522	78 069	402 079

## VI. CONCLUSIONS

In the paper, we proposed an exact algorithm and a heuristic for an  $NP$ -hard time-dependent scheduling problem with linearly deteriorating jobs, arbitrary precedence constraints and the maximum cost objective. We also presented preliminary results of numerical evaluation of the algorithms.

The main future research goal is to introduce in the proposed heuristic algorithm such improvements that will increase the quality of generated schedules and make it more satisfactory. It is also worth to check, which one of known algorithms generating all topological sortings of a given acyclic digraph will be the most effective in the exact algorithm.

## REFERENCES

- [1] S. Gawiejnowicz, *Time-Dependent Scheduling*, Springer, Berlin-Heidelberg, 2008.
- [2] S. Gawiejnowicz and B. M. T. Lin, *Scheduling time-dependent jobs under mixed deterioration*, Applied Mathematics and Computation, 216 (2010), no. 2, 438–447.
- [3] D. E. Knuth, J. L. Szwarcfiter, *A structured program to generate all topological sorting arrangements*, Information Processing Letters, 2 (1974), no. 6, 153–157.
- [4] A. Kononov, *Single-machine scheduling problems with processing times proportional to an arbitrary function*, Discrete Analysis and Operations Research, 5 (1998), no. 3, 17–37 (in Russian).
- [5] E. L. Lawler, *Optimal sequencing of a single machine subject to precedence constraints*, Management Science, 19 (1973), no. 5, 544–546.
- [6] M. L. Pinedo, *Planning and Scheduling in Manufacturing and Services*, Springer, New York, 2005.
- [7] J. B. Wang, C. T. Ng, T. C. E. Cheng, *Single-machine scheduling with deteriorating jobs under a series-parallel constraint*, Computers & Operations Research, 35 (2008), no. 8, 2684–2693.